A Study of Parallel Software Development with HPF and MPI for Composite Process Modeling Simulations

D. Shires
HPC Division
Army Research Lab
APG, MD

R. Mohan*
HPC Division
Army Research Lab
APG, MD

A. Mark
HPC Division
Army Research Lab
APG, MD

Abstract

High performance and parallel computers have greatly increased the feasibility of performing large-scale analysis for engineering applications. However, it does take some effort to effectively use these high performance computing (HPC) resources. Appropriate parallel algorithms must be developed, tested, and optimized. The investment in time and resources required to accomplish this task is most often substantial. Therefore, developments should take place in a framework that is reasonably portable with minimal code re-write. In this light, we look at two portable parallel programming methodologies: High Performance Fortran (HPF) and the Message Passing Interface (MPI). These are discussed in the context of simulations for composite manufacturing processes. We describe the mathematical modeling and related implementation approaches in the two methods, highlight some strengths and weaknesses, and provide some preliminary comparisons. We also briefly present one of the current research applications in the composites process modeling arena.

1 Introduction

Even as the overall size of the Army has been shrinking, its mission has been continually expanding. The standing force must evolve to become more lethal, more survivable, and more mobile. Composite material technology provides an effective technique to reduce the weight of Army systems, thus promoting quick force projection into potentially remote areas. The use of composite materials has traditionally been considered high-risk, as the process and mechanics involved in their manufacture was not well understood. Utilization of composites is even more problematic as they are being considered for load-bearing structures. Often these parts and assemblies can become quite large. Producing large-scale consolidated

*Dr. Mohan is an employee of the University of Minnesota permanently located at the U. S. Army Research Laboratory.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2011		**00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A Study of Parallel Software Development with HPF and MPI for Composite Process Modeling Simulations	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **U.S. Army Research Lab,HPC Division,Aberdeen Proving Ground,MD,21010**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

High performance and parallel computers have greatly increased the feasibility of performing large-scale analysis for engineering applications. However, it does take some e ort to e ectively use these high performance computing (HPC) resources. Appropri- ate parallel algorithms must be developed, tested, and optimized. The investment in time and resources required to accomplish this task is most often substantial. There- fore, developments should take place in a framework that is reasonably portable with minimal code re-write. In this light, we look at two portable parallel programming methodologies: High Performance Fortran (HPF) and the Message Passing Interface (MPI). These are discussed in the context of simulations for composite manufacturing processes. We describe the mathematical modeling and related implementation ap- proaches in the two methods, highlight some strengths and weaknesses, and provide some preliminary comparisons. We also brie y present one of the current research applications in the composites process modeling arena.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**24**	

composite structures without defects represents a new challenge for future combat systems (FCS) manufacturing.

To assist process engineers, a new process simulation tool: OCTOPUS-COMPOSE (Composite Manufacturing Process Simulation Environment) has been developed. This suite, developed by the U. S. Army Research Laboratory and the University of Minnesota, provides an understanding of the resin progression behavior inside complex mold cavities filled with heterogeneous, fibrous, porous media. Process simulations reduce both the time and cost associated with manufacturing various new composite weapon systems by giving engineers a chance to optimize the manufacturing process in a virtual environment. Applications relate to technology transitions for the improved manufacture and process maturation of composite structural components in programs such as Comanche, Apache Growth, and the Joint Strike Fighter.

In many cases, parts under consideration can be effectively modeled in the framework of a sequential simulation on a workstation. However, large-scale problems in aviation and ground vehicles, with their associated physical and geometrical complexities, require scalable computational software and high performance computing to provide accurate and feasible simulations. Process modeling flow impregnation simulations utilizes novel algorithms for accurate and fast simulations in both uniprocessors and multiprocessors [1].

The advent of the multiprocessor has greatly increased the capabilities of analysis for scientific engineering problems such as composite manufacturing. However, good performance is not guaranteed by adding more CPUs to the problem. The time, effort, sophistication, and involvement during parallel software development, debugging, optimization, and the subsequent performance testing depend highly on the parallel programming paradigms employed in most of the parallel processing systems. As much as possible, it is desirable that the differences in current and future parallel processing systems (processor and memory layout, tiered and hierarchical memory structures) be as transparent as possible. This requires parallel programming paradigms and approaches to be reasonably portable across each of the parallel platforms without any or minimal code re-write.

Accordingly, compiler research groups and parallel consortiums have been looking at various ways to facilitate the entire process. Several methods have been proposed and developed. High Performance Fortran (HPF) supports the philosophy of a high-level parallel language with parallel constructs along with parallel language compilers. It is general enough to be practical on numerous systems, from symmetric multiprocessors (SMPs) to massively-parallel processors (MPPs) to clusters. Finite element computations in composite process modeling simulations inherently involve data parallelism, which exploits the fact that the same operation is to be performed on each item in a data set. The increasing level of robustness of HPF compilers support direct portability across multiple platforms with minimum code re-writes.

Another parallel approach involves explicitly parallel message passing paradigms that provide a single-program multiple-data (SPMD) programming model with explicit communication across multiple processors. The computational problem and the finite element discretizations employed permit the decomposition of the problem domain into several sub-

domains, with each sub-domain assigned to a different processor. Sub-domain computations are performed concurrently within each processor without a need for communication across the processors. Inter-processor communications are involved across sub-domain interface boundaries and when global quantities are computed. These are achieved with explicit message passing interface (MPI) calls, which are library calls, thus providing cross-platform portability.

This paper has several goals. We first discuss composite material systems, and current systems utilizing these material types. We then discuss the need for, and the development of new approaches that provide effective solutions for both serial and parallel computer systems. We then elaborate on the HPF- and MPI-based solution techniques with appropriate discussion of the strategies, performance, and effectiveness of the solutions.

2 Composites in Army Systems and Future Combat Systems

Composite materials provide a high strength-to-weight ratio, offer improved ballistic characteristics, and have the capacity to house embedded structures (sensors, antenna, etc.). These properties make composites the prime choice for wide use in future combat systems. However, the transition from metal to composite structures has been slow. Traditional systems were mostly metal. Today we are seeing hybrid systems with composite structures becoming more load-bearing. Future systems will see a reversal where new materials will take over the major percentage of system structure.

Of the various techniques to manufacture composites, liquid composite molding techniques offer the best opportunity for mass production. These techniques provide for near net-shape parts and offer repeatable manufacturing with good fiber layout control. Two widely used approaches involved resin transfer molding (RTM) and its variants such as the vacuum-assisted resin transfer molding (VARTM). Both of these techniques use fiber preforms made from stitched material (fiberglass, graphite, etc.) and a thermosetting resin that is injected into a closed or one-sided mold. Once impregnated, the resin then reacts and cures to form a consolidated composite part.

Several things can go wrong during this process. Fiber wash can occur at areas of injection, leading to warped fiber tows. Failure to achieve appropriate vacuum in VARTM may lead to a less than desired fiber fraction. Two prominent and potentially catastrophic problems are void regions, where resin fails to permeate, and resin cure before complete injection. The process simulations continue to mature, and it currently is extremely useful for predicting the later two scenarios. Simulation-based acquisition has enormous potential in this area. By simulating the manufacturing process before making significant investments in tooling, costs and redesign (the trial-and-error approach) can be kept to a minimum.

Process simulation capabilities have been and are employed in various aviation composite programs through AVRDEC/Boeing and Lockheed/Sikorsky for programs such as Comanche, JSF components and Apache Growth rotary wing structural technology demon-

strator (RWSTD) project.

3 New Algorithms

It is important for scientists and engineers to be able to obtain accurate solutions to the physical problems within a reasonable amount of computing time and resources. The total solution time for composite process modeling simulations depend on the computational algorithms. It is not only enough to have optimal parallel software design, programming models, data structures and inter-processor communication strategies. It is highly critical to develop computational algorithms, based on physically accurate representations of mathematical models that provide the required physical and engineering solutions in an optimal time. The resin impregnation physical modeling equations and the pure finite element method computational technique and its physical accuracy and computational efficiency used in our simulations is briefly discussed.

3.1 Resin Impregnation and Process Modeling in RTM

The resin impregnation and mold filling process in RTM involves the flow of a polymeric resin through a fiber network until the mold is completely filled. The manufacturing process simulations are based on macroscopic flow models which are described by Darcy's law [2] that relates the fluid flowrate to the pressure gradient, fluid viscosity, and the permeability of the porous medium. Physically, resin impregnation and flow permeating through a porous media is a free surface moving boundary problem whereby the field equations and the free surface have to be solved and tracked. In mold filling and resin flow impregnation situations such as those seen in RTM, the primary interest is in the temporal progression of the resin inside a complex, fixed-mold cavity. The geometric and material complexities of the structural components lead to diverging/merging flow fronts and race tracking effects. Eulerian fixed-mesh approaches are effective in accounting for these complexities in RTM flow impregnation process simulations.

3.2 Numerical Approaches for RTM Resin Impregnation Modeling

Computational techniques employed in RTM flow simulations for solving the pressure field and the free surface include: 1. An explicit finite element-control volume method, 2. The pure finite element method originally developed by Mohan et al. [1,3,4].

3.2.1 Explicit Finite Element-Control Volume Method

The finite element-control volume method is a common philosophy that has been traditionally employed in most RTM simulations [5–7]. In this technique, the transient mold filling problem is treated as a quasi-steady state problem leading to stringent numerical restrictions.

The pressure solution is obtained based upon the satisfaction of a quasi-steady continuity equation given by $\bigtriangledown \cdot \left(-\frac{\boldsymbol{K}}{\mu} \bigtriangledown P\right) = 0$ based on an incompressible flow field and finite element discretization of the mold geometry.

A control volume approach is employed to update and track the transient resin front. In addition to the finite element discretizing the mold geometry, each finite element node is associated with a control volume region centered around the node. The control volume region associated with a node is bound by element centroids and element mid-sides. The continually changing flow domain is determined based on the conservation of mass principle which is applied to each of the control volume regions. The fill nature of each of the control volume regions is determined based upon a parameter called the "fill factor" associated with each node representing a control volume region. The fill factor for each node represents the fill nature of the control volume region associated with a node. The fill factor is zero for an empty control volume region (domain that has not been filled with the resin). A completely filled control volume region has a fill factor of one. A resin flow front exists in the control volume regions where the fill factor is between zero and one.

The mass flux associated with each of the control volume regions is determined based on the Darcy's velocity field obtained from the pressure solution. The time increment for the advancement of the flow front and each of the quasi-steady states is computed as the minimum time required to completely fill at least one control volume region. In this approach, the transient mold filling problem is solved for both the pressure field and fill factors in steps. The mold filling process is thus regarded as a quasi-steady process even though the process is fully transient. The solution process assumes steady state conditions at each of the discrete quasi-steady time increments. The time steps are highly restricted across each of the quasi-steady steps ensuring the stability of quasi-steady state approximations and satisfaction of Courant stability conditions. Such restrictions realistically prevent composite resin impregnation simulations for large-scale structures by increasing the number of quasi-steady steps during complete impregnation. Accordingly, this solution technique makes such large-scale simulations very much impossible even on high performance computing platforms.

3.2.2 Pure Finite Element Methodology

A recently developed new approach for the resin impregnation modeling during composite manufacturing is based on a transient form of mass conservation equation, thus accurately modeling the transient physical behavior, and is called the pure finite element methodology. This technique is briefly described next.

The objective of the resin transfer mold filling analysis is to conserve and determine the distribution of the resin mass at any instant of time. As discussed earlier, in an Eulerian fixed mold cavity, the term fill factor defines the distribution and amount of resin mass present at any time inside the mold cavity. For a constant-density resin based on Darcy's flow field, the modified mass conservation equation for the resin mass at any instant of time inside a

general mold cavity forming the Eulerian domain Ω (figure 1), can be represented as

$$\frac{\partial}{\partial t}\int_{\Omega}\Psi d\Omega = \int_{\Omega}\Psi \bigtriangledown \cdot \left(\frac{\boldsymbol{K}}{\mu}\bigtriangledown P\right)d\Omega \tag{1}$$

The modified mass balance equation involves the fill factor(Ψ) denoting the filled and the unfilled regions defining the resin distribution and the pressure field. In the case of non-isothermal situations involving polymerization curing reactions, the resin viscosity μ is a function of the temperature and the degree of cure. It will vary continually during the mold filling.

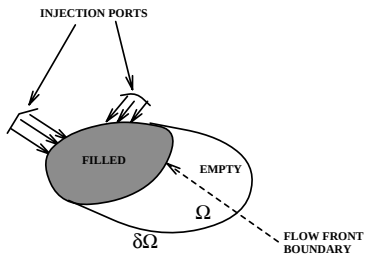


Figure 1: General partially filled mold cavity.

The resin flow inside a mold cavity (based on Darcy's flow approximation) is a pressure driven flow with the pressure gradients negligible ($\bigtriangledown P \approx 0$) in unfilled and partially filled regions, $\Psi = 1$ in completely filled regions, and the variation in the fill factor similar to those in the finite element-control volume method. The modified form of the mass balance equation involving the fill factor and the pressure field represents the governing model equation for the pure finite element formulations and is given by

$$\frac{\partial}{\partial t}\int_{\Omega}\Psi d\Omega = \int_{\Omega}\bigtriangledown \cdot \left(\frac{\boldsymbol{K}}{\mu}\bigtriangledown P\right)d\Omega \tag{2}$$

The boundary and initial conditions for the mass balance equation are normal pressure gradients at the mold wall surface, zero pressure at the resin fronts, and prescribed flow or pressure at injection ports with fill factors $\Psi(t \geq 0) = 1.0$.

Finite element discretizations for the pressure field and the fill factor field are introduced and application of Galerkin finite element methods lead to a discretized system of equations for the nodal pressure and the fill factor. These are solved in an iterative manner. The discretized system of equations is given by

$$\boldsymbol{C}\left[\boldsymbol{\Psi}^{n+1} - \boldsymbol{\Psi}^{n}\right] + \Delta t\left[\boldsymbol{K}\right]\boldsymbol{P} = \Delta t\boldsymbol{f} \tag{3}$$

where the discretized matrices are defined based on the Galerkin finite element discretizations.

The pure finite element method for mold filling analysis in RTM has significant physical, algorithmic, and computational advantages. The method computes the front location with greater accuracy. Furthermore, the computed location of the flow front at any instant of time does not depend on the discrete time step size employed to reach that stage [1]. The pure finite element methodology briefly described here does not involve the restrictions of the time step increments seen in the explicit finite element-control volume method but rather computes the position of the flow front at each of the discrete time steps which are selected by the analyst. This leads to significant reductions in the time required to complete process

simulations. These reductions are quite dramatic in the case of large-scale manufacturing process applications.

The pure finite element methodology is proven to provide a physically accurate, computationally faster, and algorithmically better solution strategy for the finite element modeling of the resin impregnation through the porous media. It has also been well demonstrated for thin, thick and large-scale composite sections [1,3,4,8]. It is to be noted that the computational advantage demonstrated by the pure finite element method when compared with the finite element-control volume technique for RTM resin impregnation simulations is solely due to the computational methodology and the algorithmic solution strategy. The effectiveness of this strategy can be seen in any computational platform, from a desktop personal computer to a high-performance computing system. The pure finite element computational methodology has also made possible large-scale complex resin impregnation simulations within a reasonable computing time. Such large scale simulations were impossible earlier. [8].

For large-scale process simulations involving high-performance computing systems, the effectiveness and computational advantages of the method is independent of the taxonomy of the parallel processor topologies (interconnection networks, memory hierarchies, etc.) when similar data structures, linear system equation solvers, programming paradigms, and communication strategies are employed in the software development. As an illustration, the normalized computational time (total solution time for the analysis) for complete impregnation of a 10-foot RAH-66 Comanche keel beam configuration involving 29,171 nodes and 58,187 elements is shown in Table 1. The computational mesh geometry employed in the simulations and the temporal resin progression contours based on representative injection locations are shown in figure 2. Our experiences with large-scale process simulations involving mesh configurations of higher order indicate that the time step increments between each of the quasi-steady states of the finite element-control volume methodology are extremely small, thereby significantly increasing the computational times and precluding the completion of large-scale process modeling simulations in a realistic time. With the computational advantage of the pure finite element method well established [1,3,4,8], the technique forms the basis of current scalable, parallel software developments.

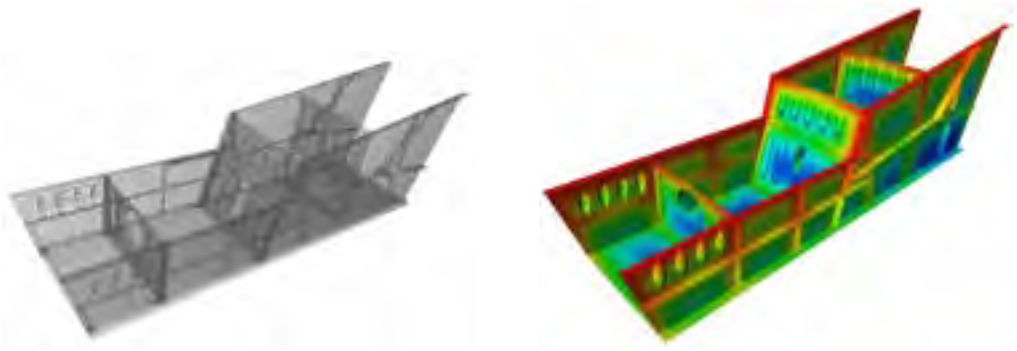


Figure 2: Process simulations: 10-foot keel beam section

Table 1: Comparison of computational times: 10-foot keel beam section.

Method	Computational time[a]
Explicit FE-CV	31.28
Pure implicit FE ($\Delta t = 0.5$ sec)	1.00

[a] Relative to the actual computational time corresponding $\Delta t = 0.5$ sec.

4 Parallel FEM with High Performance Fortran

The problem space is constructed through a discretization into finite elements, in which the mesh configuration is built up of nodes (points in three-dimensional space) and elements (formed by connection of the nodes). The computational data structures involved in the parallel software implementation consist of both elemental and nodal level structures. However, a coupled interdependency exists between the nodal and element level computations. This inherently leads to some level of communication across memories potentially residing in various processors during parallel finite element implementations. The level and cost of such inter-processor communications depend upon the employed programming models, parallel data structures, and computing architecture.

Although HPF has historically been considered more in the context of grid-based computation, it can also be used to achieve parallel solutions to problems using unstructured finite element implementations as well. Of the various parallel programming models available, HPF remains general enough to be applicable to various architectures. This highlights the flexibility of the language in an environment that has increasingly seen data parallel computers replaced by coarse-grain architectures. It also attempts to address parallelism at a high level, providing a wealth of research potential into compiler construction and methodology for high performance computing systems. The particular discussions in this paper involve the influence of data locality, language definition, and compiler maturity for inter-processor communications in actual numerical simulation codes.

HPF is a language that requires a robust standard and good compilers to achieve its maximum potential. However, regardless of low-level compiler efficiency, a significant level of data locality control exists in the hands of the parallel software developer. Various issues related to inter-processor communication found in finite element implementations, such as gather and scatter operations, and the current state-of-the-art of HPF compilers are discussed. Inter-processor communication (as well as intra-processor communication) is a major issue, and by appropriately reconfiguring the finite element data, one can achieve improved data locality. We discuss several topics, including strategies and issues related to improving the data locality, influences on the cost and size of communications, and our experiences based on parallel HPF finite element implementations. Though the engineering application employed in our study is based on composite process modeling simulations, the techniques

and the strategies apply across several disciplines and in all situations involving unstructured finite element computations.

4.1 Data Parallelism

Data parallel approaches provide a direct mathematical based framework for quick, direct parallel implementations. Data parallelism exploits the fact that often the same operation is to be performed on each item in a set of data. A data parallel program is a sequence of such operations. Currently, HPF is the most widely used language to represent data parallelism. It augments Fortran 90, which provides constructs to represent concurrent execution but not domain decomposition.

HPF achieves efficient parallelism through a combination of concurrency and locality of data reference [9]. While the two are interrelated, it is possible to discuss them separately. Concurrency assures that all processors are busy, while locality limits the potential amount of communication found in the concurrent statements. For example, a simple statement such as `A = B * C` can proceed concurrently with or without communication required between the processors, depending upon how the data were distributed.

A limited number of commercial and research HPF compilers is available today. One of the most popular and robust in support of numerous supercomputer systems is one from the Portland Group. This compiler is available on numerous Department of Defense HPC assets and is used in our current implementations.

4.2 Locality

As with any parallel code, a paramount concern involves limiting as much as possible the amount of communication that must occur between the processors in the parallel pool. Determining a program's optimal distribution of data objects operated on is a global optimization problem, and as such is not generally possible with compiler technology alone. Accordingly, HPF provides directives for data mapping (i.e., alignment and distribution to advise the compiler on how to best distribute data elements to the parallel processors. As would be expected, these directives work best at reducing communication in an environment comprised of regular, grid-based data. For instance, with a two or three-dimensional grid of data, it is relatively straightforward for the compiler to distribute data evenly across the parallel processors. For "ghost points," those items on the data mapping borders which are shared between processors, it is also feasible for the compiler to vectorize and agglomerate the data that must be communicated between processors, thus reducing the overall time spent in communication. Efficient scheduling in these cases is also possible to hide memory hierarchy latency. However, unstructured finite element mesh (FEM) configurations do not produce such regular data sets.

Communication is implicit in HPF compared to the explicit calls found in message-passing codes. While this, in principle, is a factor to make coding in HPF easier than traditional message-passing languages, it also represents an area that requires special attention if HPF codes are to perform as well as their message-passing counterparts. Communication in HPF

results from the interplay between the program being executed and the data layout resulting from the distribution directives. An obvious source of communication is found in collective operations, such as summation reduction. These operations obviously require some cross-processor communication. Furthermore, with unstructured finite element meshes, there is the distinct possibility that the HPF data mapping directives will not keep the data as processor-local as possible.

4.2.1 Collective Operations

The two most interesting (and used) collective routines in FEM implementations are gather and scatter. These collective operations are relatively straightforward in HPF. There are no default library routines in HPF to do gather operations. Rather, the operation can be achieved through a small bit of code. The HPF code to do a gather operation is comprised of nested `INDEPENDENT do` loops. However, there is a `SUM_SCATTER` HPF library function to perform the required reduction. In this, the full details of the implementation are often proprietary and architecture specific, but most likely this call computes a communication schedule for data going to and arriving from remote nodes, moves the data, and then computes the reduction. Robust compilers can also perform the reductions locally before sending out the data.

These communication operations can be very expensive. A profile of our code revealed that it was communication-bound, with well over 50% of the execution time being spent in calls to the `SUM_SCATTER` library routine. Approximately 20% of the time was spent in code segments performing gather operations. The library routine `SUM_SCATTER` is called repeatedly, even for small problems. Since it is a library routine, our assumption was that each time it was called, a schedule was being computed and executed, and any information gathered by the scheduling algorithm was being discarded before the next call. While the details of the communication computation are hidden, it is easy to envision a less than optimal scheduling algorithm taking at least $O(n^2)$ time with n being the number of elements in a finite element mesh. Our conversations with the Portland Group confirmed that the schedules were being computed repeatedly.

Obviously, the ability to reuse communication schedules is essential to achieving good performance with codes that repeatedly use scatter and gather operations. Although the ability to reuse schedules is not specified in the current HPF standard, the Portland Group has provided a vendor-specific patch that will allow communication schedules to be reused, at least in scatter operations. The schedule can be called repeatedly, hence eliminating the need to recompute the schedule at each call to `SUM_SCATTER`. The Portland Group reports that some users have experienced a three-fold code speedup after switching to reusable schedules. We did note a marked decrease in execution time with reusable schedules. The execution time for one sample problem fell from 15,534.78 seconds to 6,111.37 seconds, cutting the time by a factor of 2.5.

Equally problematic are gather operations that generate volumes of code to compute off-node data locations. The ability to reuse communication schedules inside of `INDEPENDENT do` loops used for gathers is also under investigation. To accomplish this, syntax has been

proposed by the Japan Association for HPF (JAHPF) [10]. The process involves using a `INDEX_REUSE` directive to avoid costly repeated schedule computations. The Portland Group is planning on incorporating many of the features proposed by the JAHPF. HPF retains most of its popularity in Europe and Asia. As a result, much of its future is being developed there. The language definition will undoubtedly continue to evolve to address and incorporate these changes that currently reside in the vendor-specific realm.

4.2.2 Mesh Reconfiguration

The locality issues previously highlighted rely on the sophistication and the maturity of the parallelizing compilers. From a parallel finite element code user perspective, significant improvements in data locality, thus limiting interprocessor communication and reducing the execution time, can be achieved by optimal reordering of the nodal and element mesh configuration data. The node and element numbers are reordered such that there is limited interprocessor communication between the data after the mapping of the data to the processors.

As previously stated, locality of reference greatly impacts the performance of a data parallel program. HPF provides several directives and distributions to map data and promote locality. The most common of these are the `DISTRIBUTE` and `ALIGN` directives. The `DISTRIBUTE` directive indicates how an array is partitioned to the various processors. Array alignment, to make sure that corresponding entries are on the same processor, can be specified using the `ALIGN` directive. The array dimensions can be distributed as `*`, `BLOCK`, or `CYCLIC`.

For some applications, such as two-dimensional image processing routines, these distributions map easily and intuitively to the data to promote reference locality. However, for unstructured finite element mesh-based data, we usually deal element- and node-based data sets. Depending upon the quality of the original finite element mesh, a `BLOCK` or `CYCLIC` distribution of the data will require differing amounts of communication. For example, consider the `BLOCK` distributions of the nodes and elements arrays in Figure 3. This figure shows how the element distribution across processors can require extensive communication depending upon how the elements reference the nodal-based data.

It should also be noted that rarely will even "good" meshes be optimal in all cases. Varying the number of processors alone with the same mesh will cause differing amounts of communication. For example, consider a linear triangular finite element e consisting of nodes *n1, n2,* and *n3* to form the element e in a computational mesh. Assume that there are 10,000 elements, 4,000 nodes in the computational mesh, and $e = 1$, *n1* $= 1$, *n2* $= 2$, and *n3* $=$ 1,900. Further assume that all the data is distributed in block format. If we used two processors, the first 2,000 nodes and 5,000 elements would be local on processor 1. So, no communication would be required for a computation based on element e requiring any nodal data. However, if we were to use four processors, node *n3* data would now reside on processor two, rather than on processor one, and would then require some communication.

Accordingly, it is extremely advantageous to have a pre-processing step decomposing the data in a smart fashion based on the expected number of processors used for finite ele-

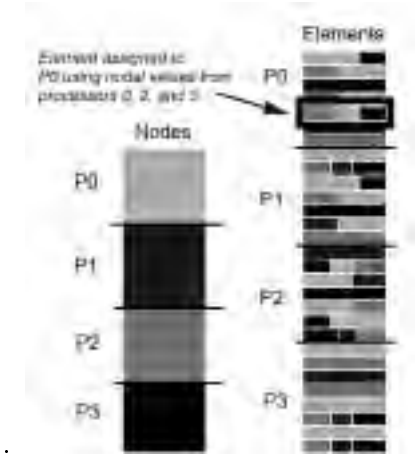


Figure 3: Original mesh configuration with high communication requirements.

ment computations employing HPF-based parallel software developments. This technique is already used in many SPMD message passing-based implementations. In this, the computational domain is decomposed into a number of partitions equal to the number of processors. A similar strategy can be employed in HPF parallel software model executions.

First, the original computational mesh is partitioned using an unstructured graph- or mesh-partitioning software such as Metis [11]. These graph-partitioning tools attempt to divide the mesh, either according to the nodal or element data, into a number of partitions while attempting to limit the number of shared nodes between partitions. This gives us a list of elements for each domain. Second, we compute the domain shared node vectors. Third, for each domain or partition, we group the nodes that are shared between domains and renumber them first. For example, for partition 0, we renumber all the nodes shared with partition one, then partition two, etc. This step helps promote data agglomeration and message vectoring. By placing all of the shared items in a contiguous location, the compiler will just have to do one send operation. On a shared or distributed-shared memory platform, this will consist of a memory starting location and vector length. Finally, we renumber all of the domain-interior nodes. We then proceed to the next domain. This process, if implemented efficiently, can be very fast. The renumbering technique employed is on the order of $O(n\ log\ n)$, where n is the number of nodes that are shared between the various domains. Our implementation is hence bounded only by the speed of the mesh partitioning software. Figure 4 shows how nodal renumbering in the elements can reduce the required communications across processor memories. Communications now consist of nodal-based data shared between the domains and artifacts left over from domain partitions that cannot exactly match the `BLOCK` or `CYCLIC` distribution boundaries.

This very minimal extra pre-processing of the original data has resulted in reduced execution times in every case. In some cases, the payoff has been outstanding. Consider the results displayed in Table 2. This table shows the time for an analysis using a mesh with 29,171 nodes and 58,187 elements on the Cray T3E computer. Note that the time for a 16

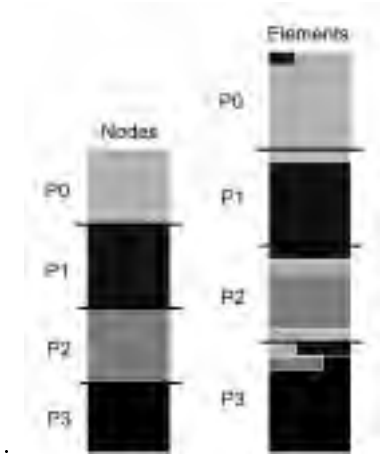


Figure 4: A better mesh configuration realized by renumbering.

Table 2: Execution times (seconds) of an unstructured finite element analysis on a Cray T3E-1200.

Problem		Number of processors				
		2	4	8	16	32
Airframe structure	Original mesh	20951.89	13117.40	8515.41	6111.37	3603.69
	Renumbered Mesh	19189.21	9989.31	5553.45	3135.03	2095.47

processor simulation (6,111.37 seconds) was cut to 3,135.03 seconds, or roughly in half. We are currently assessing the impact of the agglomeration and vectoring steps to see if they are providing any increased efficiency realized by the compiler.

In addition to the reduction in execution time, the size of the data communicated and the number of sends/receives are significantly reduced. Figure 5 shows how this strategy significantly reduced the total amount of data that had to be communicated between processors. It should be noted that the total send and receive requests were also reduced, not just the total amount sent.

While the aforementioned techniques are very effective at reducing communications, they still do not address the problem of using a computational mesh that cannot be rigidly broken along domains to fit into the compiler partition sizes. `BLOCK` distributions result in symmetric block sizes of size $\frac{n}{p}$ where n is the cardinality of the data set and p is the number of processors. This distribution pattern will create ghost point communication because the mesh partitioning step often produces uneven partitions. One partition's node data may be slightly misaligned onto another processor. This is another detractor that should be eliminated to optimize the code, and provide more equitable comparisons with message-passing implementations.

One possible way to eliminate these communications is through "mesh padding." In this process, we determine a graph partition and renumber as before. Then, we compute the block distribution boundaries based on the mesh size and compare how well the graph/finite

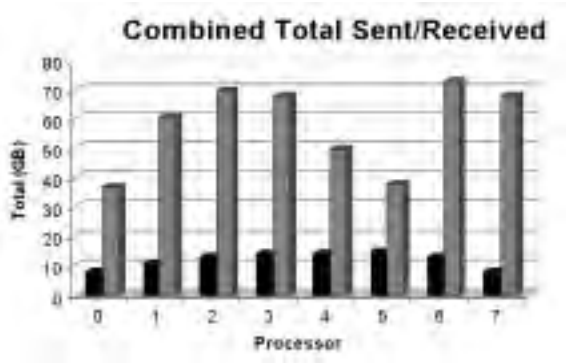


Figure 5: Total sends/receives (gray–original, black–renumbered) reduced through mesh renumbering.

element partition maps to these block boundaries. We then create nodes and elements as needed to make the finite element mesh fit the partition. Depending upon the initial mesh configuration, the number of nodes and elements that have to be created can be quite high. This approach has several drawbacks. The original mesh intent may be lost with the addition of new nodes and elements. Furthermore, we are reducing communication at the expense of added computations.

A better approach has recently been developed. The latest Portland Group HPF compiler (version 3.0) includes an asymmetrical block distribution and eliminates the need for mesh padding. This approach allows for "soft" block boundaries that can be set by the software at runtime and not set by the compiler at compile time. Combined with the mesh renumbering technique, this should limit all communications except for true domain boundary data. The syntax is:

```
!HPF$ DISTRIBUTE A(GEN_BLOCK(DIST)).
```

Here, `A` is the array to be distributed and `DIST` is an array whose cardinality is the number of processors on which the code will run. Each element of `DIST` contains the size of the data block that is to be placed on the processor. Since we have all pertinent data from the renumbering step, we should be able to provide precise numbers for the `DIST` array, thus boosting performance even more. Figure 6 shows the configuration of this optimal data-parallel mesh.

Unfortunately, being on the cutting edge has its faults as we were unable to utilize this new feature. As released, the compiler is unable to generate an executable code that uses generic block distribution and is compiled with shared memory addressing mode enabled. In the case of the T3E, shared memory referencing causes the compiler to generate code that uses very efficient Cray one-sided get and put communication calls [12].

After disabling shared memory compilation to use generic blocks, the compiler began to replicate arrays that were used in gather loops. The arrays used in gather loops were not conformable (not uncommon for element and node arrays), and rather than generate potentially costly communication runtime library calls, the compiler generated code that

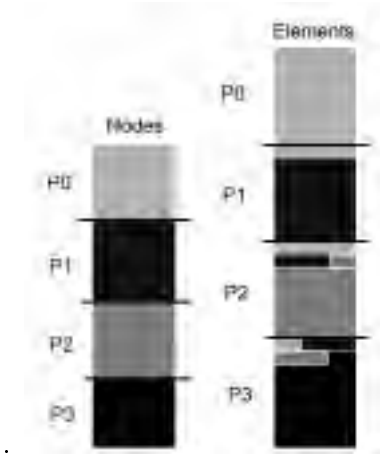


Figure 6: The optimal mesh configuration combining renumbering and asymmetric block sizes.

would replicate one of the arrays to each processor every time the gather was executed. In contrast, shared memory referencing relaxes the compiler rules and does not cause this replication. Rather, the compiler generates in-line calls to optimized Cray communications libraries. Very limited testing was required to determine that the cost of replication would well exceed any benefit from generic blocks. We will revisit this issue as the compiler is updated to support both approaches.

5 Parallel FEM with Message Passing Paradigms

The message passing interface (MPI) is a library-based explicitly-parallel programming methodology. It consists of code that is instrumented with calls to a runtime library for interprocessor communication. These routines are callable from the various Fortran implementations, C, and C++. The library is robust, with over 100 different routines. It is also simple in many ways, with many large parallel codes only using about 6 different routines for basic communication.

As it stands, compilers do not optimize for MPI usage. It is entirely up to the developer to achieve good performance by being creative in MPI usage. Accordingly, MPI is often known as the "assembly language" of parallel computing. The attention to detail required is often meticulous.

MPI uses the single program-multiple data (SPMD) approach to achieve parallelism. This means that a single program image is spawned across many processors, each operating on (hopefully) local data. SPMD programs are not systolic. In other words, the program counter can be at different locations in each copy of the running program. Also, the processes can execute different statements (and in effect different programs) by using conditional branches based on a unique processor identification number.

The mostly local nature of computations involved in the finite element techniques makes

it easily possible to use SPMD type of parallel programming with explicit message passing paradigms. The finite element computational domain is decomposed into a number of sub-domains based on the number of processor partitions involved for problem execution. Considerable research have been performed over the years for developing efficient graph partitioning techniques and software with the objective of reducing the number of interface nodes and elements across each of the partitioning domains. It is also possible to distribute the domain partitions based on the computational loads involved in different parts of the problem domain. The approach that is employed in this work for parallel software development is based on the non-overlapping finite element domain decomposition. In this the computational domain is split into a number of sub-domains based on the number of processors. The finite element boundaries of any given sub-domain coincide with the sub-domain boundaries with only the interface nodes being part of the different sub-domains. The finite elements that make up each of the sub-domain are completely local to the sub-domains. Issues relating to parallelism in message passing interface (MPI), compiler capabilities and programming strategies are briefly discussed in subsequent sections. The non-overlapping finite element mesh decompositions have been obtained using the graph partitioning software Metis and/or ParMetis [11] in our parallel software developments.

From finite element software development perspective, the parallel software developer is more involved during the development of explicit message passing parallel code. Careful thought should be given to various aspects including the setting up of the problem, boundary conditions, equation system solver, thus increasing the difficulty factor during the parallel software developments. Few of these issues are briefly highlighted in a later section.

5.1 Parallelism in MPI

Unlike HPF, which attacks parallel programming from a higher conceptual level, MPI is known as explicit parallel programming since the software developer must instrument the code with appropriate communication calls to achieve the desired parallelism. This has both good and bad connotations. Well written code using few synchronous communications and blocking calls can be very efficient. Conversely, the compiler cannot optimize poorly structured communication since this is done strictly through user calls to library routines. MPI most closely associates with the Single Program-Multiple Data (SPMD) model of parallel computing. Each processor has a copy of the program with local storage space for variables. There are no shared variables amongst the processes. Each process works with unique data, and all variables and data are communicated by explicit calls to a runtime message passing library. MPI is probably the most widely used parallel programming method today, and as such is very portable across platforms [13].

Domain decomposition, the partitioning of data to allow for parallel execution, is done through directives in HPF codes. MPI provides no such functionality. Instead, the user must partition data sets in a way to limit communication requirements between the various sub-domains. This partitioning problem is in no way trivial. The graph partitioning software such as Metis and ParMetis [11] provide this type of functionality.

5.2 Compiler Capabilities and Programming Strategies

Current parallel code developments are within the framework of Fortran 90. Fortran 90 language is newer than the FORTRAN 77 standard and contains several new constructs and features. The compilers for Fortran 90 have currently been known to produce less efficient code than their FORTRAN 77 counterparts. However, Fortran 90 compilers continue to mature and improve in efficiency with better back end code generators. Furthermore, the language contains several features, such as abstract data types, to promote quick code expansion.

As previously stated, use of explicit message passing requires the software developer to explicitly communicate data between the processes running on different processors. Domain decomposition is being normally done "off-line,", with calls to partitioning algorithms such as Metis for setting up of the problem data for each of the processors. Hence, pre and post processing are complicated by the developer having to create code ripe with conditional branches for initial data distribution and result collection at the end. Also, the use of blocking receives and barriers should be limited. Furthermore, domain data interchange should not be based on a fixed schedules. For instance, code should not be written where process 1 waits for data from process 2, then process 3, etc. If process 2 is lagging and is not ready to send, n communication calls from the other processes will be delayed and unable to complete while waiting for process 2. In this case, process 1 should be able to receive from any process ready to send. Use of MPI "wildcards" in communication can be used as much as possible to preclude these situations and allow for better runtime load balancing [14].

Since each process has its own copy of the code and corresponding stack, care should be taken to declare potentially large arrays of a size no larger than that needed by the individual process. However, optimizing for parallel execution falls naturally from serial optimization since multiple copies of the code will be executing.

5.3 Issues from Finite Element Perspective

Due to the explicit partitioning of the computational domain and with each processor and sub-domain having its own set of data and executing its own copy of the code, careful consideration is needed during any parallel finite element software development. Few of the issues are briefly highlighted next.

The domain decomposition leads to number of interface nodes shared among many sub-domains. Normally, in finite element implementations, the external conditions are given based on nodal or elemental information. For example, the injection conditions in composite process modeling simulations can be either Dirichlet or Neumann type. If a Neumann type condition is present at any of the interface nodes, which could be shared by more than one processor or belong to more than one sub-domain, the software developer must carefully account for such possibilities, so that external Neumann type loading conditions are not duplicated at the common interface nodes. On the other hand, a state variable such as fill factor in the composite process modeling simulations has the same value for each of the shared interface nodes, even though they may be shared by different processors. Similar

considerations apply for the Dirichlet type boundary conditions. This is one example of carefulness that is needed during parallel software development employing explicit message passing paradigms, and adding to the difficulty factor, including the time and effort during parallel software development. On the other hand, HPF approaches provide a direct mathematical based framework for quick, direct parallel implementations.

5.3.1 Local and Assembled Matrices and Vectors

In explicit message parallel algorithms, finite element matrices and vectors are local to each of the sub-domains and the processor. Each processor performs operations on its own part of vectors and matrices. In the case of non-overlapping domain decomposition, such as the one employed in this work, sub-domain boundaries coincide with finite element boundaries, i.e, each finite element belongs to one sub-domain only. Figure 7 illustrates a simple domain partitioned into four sub-domains. The finite elements are local to each of the sub-domains. However, the nodes are shared with neighboring sub-domains and the nodes can be classified into three different classes as noted in the Figure 7. There are interior nodes I, which are completely local to the sub-domain, interface(sub-domain boundary) nodes SB and neighboring sub-domain nodes NB, which have the same geometric location as the interface nodes SB, but belong to the adjacent sub-domains and processors. The communication across sub-domains take place as shown by the arrows.

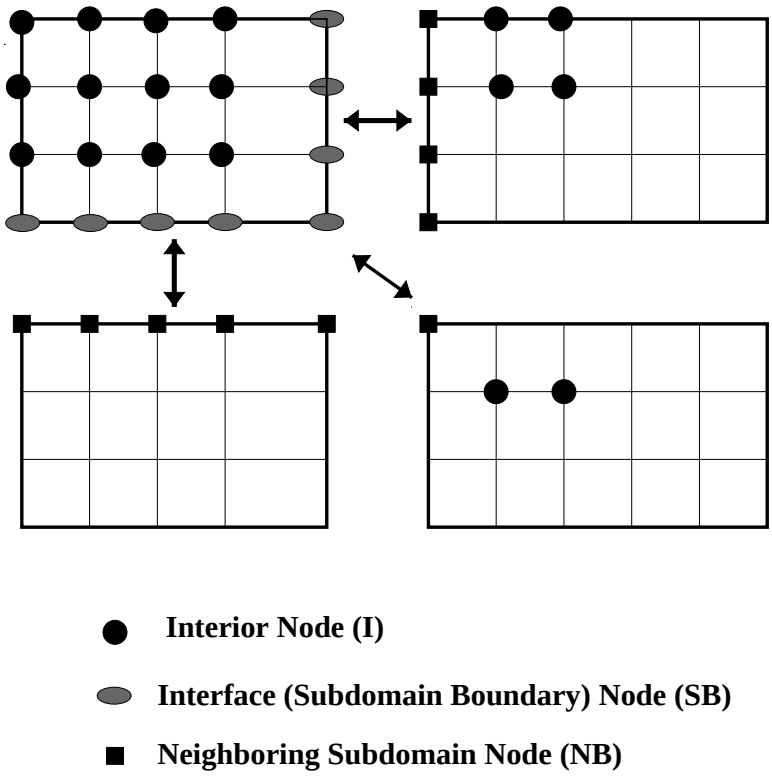


Figure 7: A simple domain decomposed into four sub-domains

During finite element computations, both computed sub-domain matrices and vectors can be either in local or in an assembled form in each of the processors. Each processor is locally responsible for its own data. An assembled vector contains full entries for both the interior and boundary nodes. In a finite element perspective, the assembled form of a vector contains the assembled contribution from both its local contributions and contributions from the neighboring sub-domains and processors it shares with. A local vector contains only the sub-domain contributions. Thus, a local vector contains full values for interior nodes and only partial values from a sub-domain for interface nodes. The use of the local and assembled entries makes it easier for the correct computation of the vector-vector inner products during the iterations of an iterative solver such as the preconditioned conjugate gradient that has been used in this work and described in the next section.

During the computations in an iterative solver, it is necessary to evaluate the inner product of two vectors to determine the solution norm. The inner product is required for the entire computational domain, as the solution norm is based on the entire computational domain. However, the vectors belong to each of the sub-domains and the contributions to the global norm from each of the processors have to be added to compute the correct norm. If the assembled vectors in each of the sub-domains are employed to determine the sub-domain contribution to the inner product, then the vector components for the interface nodes are mentioned several times and the result will be incorrect. One solution to produce the correct result is to divide the inner product component for $u_i v_i$ for the interface nodes by a 'repetition factor' for this node. The 'repetition factor' is the number of times a particular node appears in all the sub-domains, which is easily obtained from the graph partitioning phase. The inner product result can also be obtained correctly if one of the vectors is in an assembled form and the other vector is in local form.

MPI communication calls are used to determine the assembled entries for the sub-domain boundaries. For any sub-domain, vector entries u_{SB} corresponding to the interface boundaries are known. This is then sent to the neighboring sub-domains using SEND MPI communication calls. At the same time each processor receives the contribution u_{NB} from each of its neighboring sub-domains. This is achieved through RECEIVE MPI communication calls. The neighboring contributions are then assembled to the local domain vector to form the assembled vector $\hat{u}$ ($\hat{u} = u + u_{NB}$). MPI_ALLREDUCE communication calls are then used to determine the global inner product from the sub-domain inner product computations. Send and Receive operations are shown by arrows in Figure 7. The interface nodal assembly operations performed are standard computational procedure in finite element computations.

5.3.2 Linear System Solver

The composite manufacturing process simulations involve the solution of linear system of equations based on a sparse, symmetric positive definite matrix A of the form:

$$Ax = b$$

An iterative conjugate gradient algorithm following the developments from K. H. Law [15] for MIMD computers is employed here. A similar procedure has been followed and extended

for a preconditioned conjugate gradient algorithm in [16]. The matrices and vectors refer to the sub-domain level. As discussed earlier, assembled and distributed vector forms are employed. Assembled forms are denoted by $(\hat{\cdot})$. The iterative procedure for the conjugate gradient iterative algorithm is presented next. The superscript SD refers to the sub-domain vector and matrix quantities.

- $\{x_0^{SD}\} = 0$, $\{r_0^{SD}\} = \{b^{SD}\}$
- Do $i = 0, 1, \ldots\ldots$ until convergence

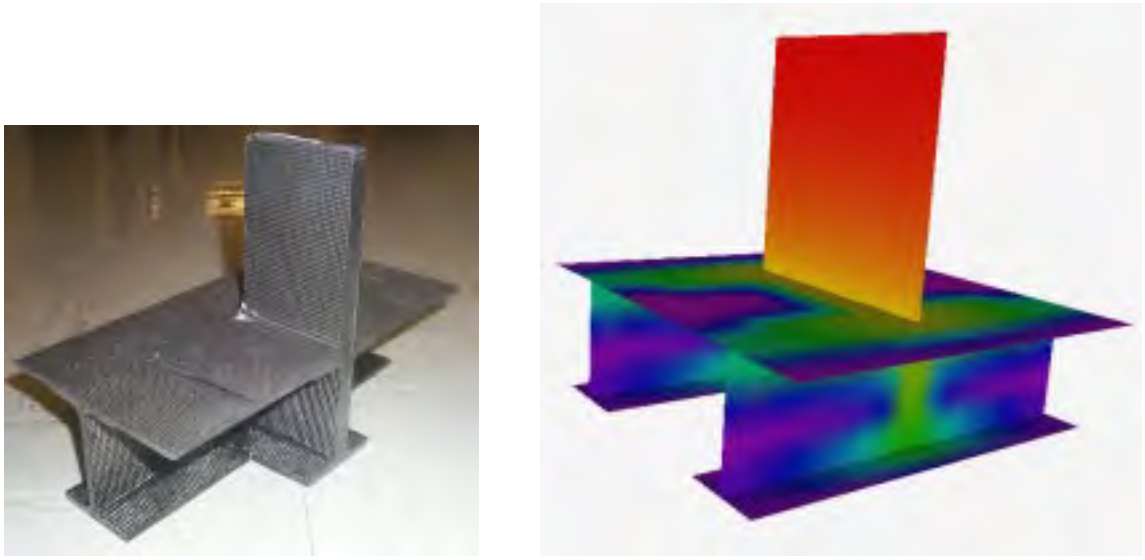
  1. If $i = 0$ Go To Step 5
  2. $\{u^{SD}\} = \left[A^{SD}\right]\{p^{SD}\}$
     $\beta^{SD} = \{p^{SD}\}^T\{u^{SD}\}$
     $\sigma_{SD} = \beta^{SD}/\gamma$
  3. MPI_ALLREDUCE $\frac{1}{\alpha} = \Sigma\sigma^{SD}$
  4. $\left\{x^{SD}\right\} = \left\{x^{SD}\right\} + \alpha\left\{p^{SD}\right\}$
     $\left\{r^{SD}\right\} = \left\{r^{SD}\right\} - \alpha\left\{u^{SD}\right\}$
  5. SEND $\{r^{SB}\}$; RECEIVE $\{r^{NB}\}$
     $\left\{\hat{r}^{SD}\right\} = \left\{r^{SD}\right\} + \Sigma\{r^{NB}\}$
     $\rho^{SD} = \left\{\hat{r}^{SD}\right\}^T\left\{r^{SD}\right\}$
  6. MPI_ALLREDUCE $\gamma_{new} = \Sigma\rho^{SD}$
  7. If $(i = 0)$ Go to Step 9
  8. If $\left(\frac{\gamma_new}{\gamma} < \epsilon\right)$ STOP
  9. $\left\{p^{SD}\right\} = \left\{\hat{r}^{SD}\right\} + \frac{\gamma_new}{\gamma}\left\{p^{SD}\right\}$
  10. $\gamma = \gamma_{new}$

In the above iterative solution algorithm, A is the system matrix; b is the right hand side; x is the unknown solution; r is the residual vector;p is a temporary working vectors, ϵ is the specified error tolerance. Assembled sub-domain vectors are denoted by a hat. For example, r is a local vector in a sub-domain and $\hat{r}$ is the same sub-domain vector in the assembled form. The multiplication of a local matrix by an assembled vector is a local vector. The transformation from local to assembled vector requires communication of the interface data between neighboring sub-domains and then to assemble the received data to the local vector. Each iteration of the conjugate gradient algorithm contains a matrix vector product and two inner products; one communication operation per iteration between neighbors and two reduction operations are necessary to obtain the cumulative value of the scalars $\frac{1}{\alpha}$ and γ from their sub-domain contributions on all the processors. As before, the matrix vector products have been computed based on element level matrices avoiding any assembly operation of the sub-domain matrices. Since sub-domain matrix assembly operations do not involve any

communications, it will be interesting to perform comparisons based on assembly of the sub-domain matrices in a sparse storage format and performing the sub-domain level matrix vector products. Based on the sparse system, this adds additional storage requirements.

6 Applications and Analysis Results

Process modeling simulations have been applied to various composite structural developments and are playing a major role in various composite affordability initiatives towards process maturation. Figure 8 shows for illustrative purposes one such application of process modeling simulations to a risk reduction version of the Comanche gear box assembly. The transient resin impregnation behavior modeled by the process modeling simulations compared and modeled well the actual behavior during the prototype manufacturing. More importantly, the process modeling simulations will provide the critically needed modeling and simulation technology when actual full size composite structural components are manufactured. The risk involved in obtaining a good working production methodology for large-scale full size composite structural components is very high. By appropriately modeling and simulating the various injection and processing parameters virtually process maturation can be obtained, thus impacting the composite affordability. These process simulations also provide the necessary framework for a simulation based acquisition strategy for the manufacturing and acquisition of composite structural components in both defense and civilian applications.



(a) Part section

(b) Resin progression contours

Figure 8: Resin impregnation in Comanche gear box assembly

One of our key objectives in this work is the comparison of the parallel software de-

velopment process, intricacies and performances between the parallel programming models thorough high performance fortran (HPF) and message passing passing interface (MPI). From that perspective, we have presented and discussed earlier our current experiences in this paper. The parallel software codes for the composite process modeling simulations described earlier have been developed, tested and validated using both the data parallel HPF programming language paradigm and explicit message passing paradigm (MPI) paradigm. We are currently in the process of generating the comparison data based on the total execution time, computational cost, communication cost and difficulty factor for a typical application program based on controlled simulation runs using the developed codes under these two paradigms. Preliminary MPI based times have been obtained and are not presented here. Further optimization, refinements and studies are currently being conducted on the present code developments. Recommendations on the use and practicality of these two paradigm approaches for parallelization efforts of existing and new codes will be presented based on our experiences in subsequent publications.

7 Concluding Remarks

The importance of computational algorithms and their role in large scale composite manufacturing process simulations was briefly highlighted. The objective of the parallel software and parallel processing systems is not only reducing the time it takes to solve a particular problem, but also making possible the solution of large-scale problems that were not previously possible. The effectiveness of a recently developed implicit transient pure finite element algorithm towards this goal for resin impregnation modeling was briefly presented and demonstrated. Various issues and experiences during parallel software development for composite manufacturing process simulations employing two common parallel programming paradigms (High Performance Fortran (HPF) and Message Passing Interface (MPI)) were described.

These discussions, though presented in the context of composite manufacturing process simulations, are common and directly applicable to parallel software developments and applications involving unstructured finite element meshes. The data parallel paradigm through high level languages such as HPF and parallel compilers provides a direct mathematical based framework for direct parallel implementations. Such approaches may be effective in certain applications, where the motivation is to obtain a reasonably performing, parallel, scalable, portable code in a short cycle time, within the constraints of the availability and maturity level of the compiler. This is especially true if the parallel software may have limited usage, from cost and business considerations. The explicit message passing approach, though expected to yield higher performance levels, have an increased level of difficulty during software development and require a higher level of sophistication.

8 Acknowledgments

This research was made possible by a grant of computer time and resources by the Department of Defense High Performance Computing Modernization Program. The authors also with to thank Douglas Miles of the Portland Group for his assistance in providing immediate access to the latest (and sometimes developmental) HPF compilers. Dr. Mohan acknowldeges the support from University of Minnesota and the Army High Performance Computing Research Center at University of Minnesota.

References

[1] R. V. Mohan, N. D. Ngo, and K. K. Tamma. On a pure finite-element-based methodology for resin transfer mold filling simulations. *Polymer Engineering and Science*, 39(1), January 1999.

[2] H. Darcy. *Les Fontaines Publiques de la Ville de Dijon.* Delmont, Paris, 1856.

[3] R. V. Mohan, N. D. Ngo, K. K. Tamma, and K. D. Fickie. On a pure finite element based methodology for resin transfer mold filling simulations. In R. W. Lewis and P. Durbetaki, editors, *Numerical Methods for Thermal Problems*, volume IX, pages 1287–1310, Atlanta, GA, July 1995. Pineridge Press.

[4] R. V. Mohan, N. D. Ngo, K. K. Tamma, D. R. Shires, and K. D. Fickie. Process modeling and implicit tracking of moving fronts for three-dimensional thick composites manufacturing. In *AIAA-96-0725, 34 th Aerospace Sciences Meeting*, Reno, NV, January 1996.

[5] C. A. Fracchia, J. Castro, and C. L. Tucker. A Finite Element/Control volume simulation of Resin Transfer Mold Filling. In *Proc. of the American Society For Composites, 4th technical conference*, pages 157–166, Lancaster, PA, 1989.

[6] M. V. Bruschke and S. G. Advani. A Finite Element/Control Volume Approach to Mold Filling in Anisotropic Porous Media. *Polymer Composites*, 11(6):398–405, 1990.

[7] F. Trouchu, R. Gauvin, and D. M. Gao. Numerical Analysis of the Resin Transfer Molding Process by the Finite Element Method. *Advances in Polymer Technology*, 12(4):329–342, 1993.

[8] R. V. Mohan, D. R. Shires, A. Mark, and K. K. Tamma. Advanced Manufacturing of Large Scale Composite Structures : Process Modeling, Manufacturing Simulations and Massively Parallel Computing Platforms. *Journal of Advances in Engineering Software*, 29(3-6):249–264, 1998.

[9] Charles H. Koelbel, David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance Fortran Handbook.* The MIT Press, 1994.

[10] Japan Association for High-Performance Fortran. *HPF/JA Language Specification*, January 1999.

[11] George Karypis and Vipin Kumar. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices.* University of Minnesota and the Army HPC Research Center, 1997.

[12] Portland Group, 2000. Private Communication.

[13] Peter Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann, 1997.

[14] Ted G. Lewis and Hesham El-Rewini. *Introduction to Parallel Computing.* Prentice Hall, 1992.

[15] K. H. Law. A parallel finite element solution method. *Computers & Structures*, 23(6):845 – 858, 1985.

[16] R. Kanapady. Parallel implementation of large scale finite element computations on a multiprocessor machine: Applications to process modeling and manufacturing of composites. Masters Thesis, 1998. University of Minnesota.